Phase 2

Team 24

Team Members	IDs
Abdelrahman Mohamed Ezzat	1190158
Ali Mohamed Ali Hashish	1190223
Farah Mohamed Abdelfatah	1190176
Ziad Ahmed Hamed	4200002

Project Idea:

Face and Image Cartoonization:

We offer the user the choice to either only cartoonize the face or cartoonize the whole frame captured using a video (or web-cam).

Block Diagram:

Overall Input: A video.

Overall Output: The cartoonized video.

Original Image:

The original image before any preprocessing.

This is the input to the upcoming stages.

Cartoonize Face only?:

A parameter for the user to decide whether they want to cartoonize the face only, or the whole image.

Input: A parameter stating whether the user wants to cartoonize only the face or the whole image.

Face Detection:

Detecting the face to determine which part will be cartoonized. We may use LBPH (Local Binary Pattern Histogram).

Input: The original image.

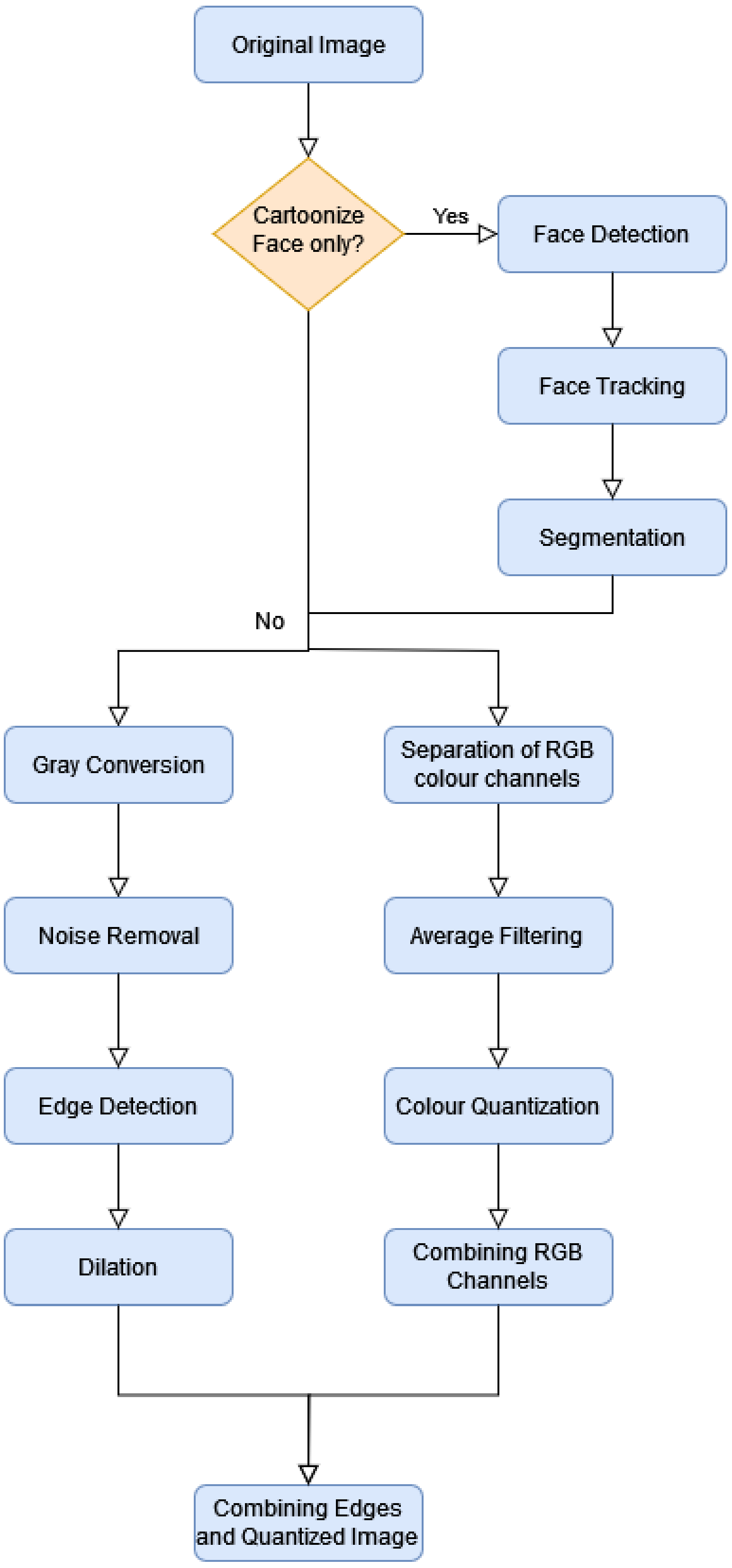
Output: The part containing the face (bounding box).

Face Tracking:

Keeping track of the face, to overlay the cartoonized image. We can use mean-shift or tracking by detection.

Input: History of the part detected from the face (the previous face detection outputs).

Output: The part containing the face being tracked.



Segmentation:

Segment the face, so that the cartoonization is applied to only the face.

Input: A bounding box of the face part.

Output: An image separating the face from the original image.

Gray Conversion:

Converting the image to gray image, to minimize the number of operations needed, improving performance.

Input: Original Image (or the detected face part of the image).

Output: Gray scaled image.

Noise Removal:

Removing unwanted noise from the image, to obtain better results. We can use Median filtering.

Input: Gray scaled image.

Output: A smoothed image without noise.

Edge Detection:

Detecting edges using canny edge detection. This will be used to enhance the lines and edges in the cartoonized image, giving it a more defined look.

Input: Image after noise removal.

Output: Image with edges.

Dilation:

Thickening the edges so that they are more apparent and clearer.

Input: Image with edges.

Output: Image with thicker edges.

Separation of RGB Colour Channels:

This is done so that the upcoming steps are performed on each channel individually, to produce more agreeable results.

Input: Original Image (or the detected face part).

Output: 3 images, one for each channel.

Average Filtering:

Smoothing the 3 colour channels to give a blurring effect. This helps in reducing colour quantities, as each pixel would be affected by its neighbouring ones.

Input: The 3 images.

Output: Blurred images.

Colour Quantization:

Decreasing the number of colours compared to the original image, since a cartoonized colour palette is not as rich as the normal one.

Input: Blurred images.

Output: Quantized colours for each channel (RGB).

Combining RGB Channels:

Combining the 3 channels again to obtain an RGB image.

Input: 3 quantized images

Output: 1 RGB image

Combining Edges and Quantized Image:

Combining the output of both branches, to obtain the final cartoonized image.

Input: Edges and quantized RGB image.

Output: Cartoonized image.

Additional Comments:

We might use Bilateral filter for noise removal instead of Median filtering. In which case this would require a non-primitive function (we will decide after comparing results).

Used Algorithms:

We implemented Viola jones for face detection. As for cartoonization, we followed the pipeline discussed above, with few modifications to some models, such as the quantization model.

We also used 3 different ways when quantizing colours

1. We quantized using this equation found in "Image Processing based Image to Cartoon Generation: Reducing complexity of large computation arising from Deep Learning" paper.

$$P = \left\lfloor \frac{p}{a-b} \right\rfloor * \left(a + \frac{b}{2}\right)$$

2. We tried Kmeans for colour, and it worked well with faces
3. We finally tried another Kmeans approach that involved dynamically calculating the number of centroids. This too produced amiable results, however it was rather slow compared to other approaches. So we decided to only use it in still pictures.

Experimental Results:

We experimented with two pipelines, one using built-in Haarcascade, and another using our own implementation of Viola Jones face detection algorithm

Haarcascade:

Results were quite good, but it only detects faces when they are facing forward. It fails to detect them if they face sideways.

Viola Jones:

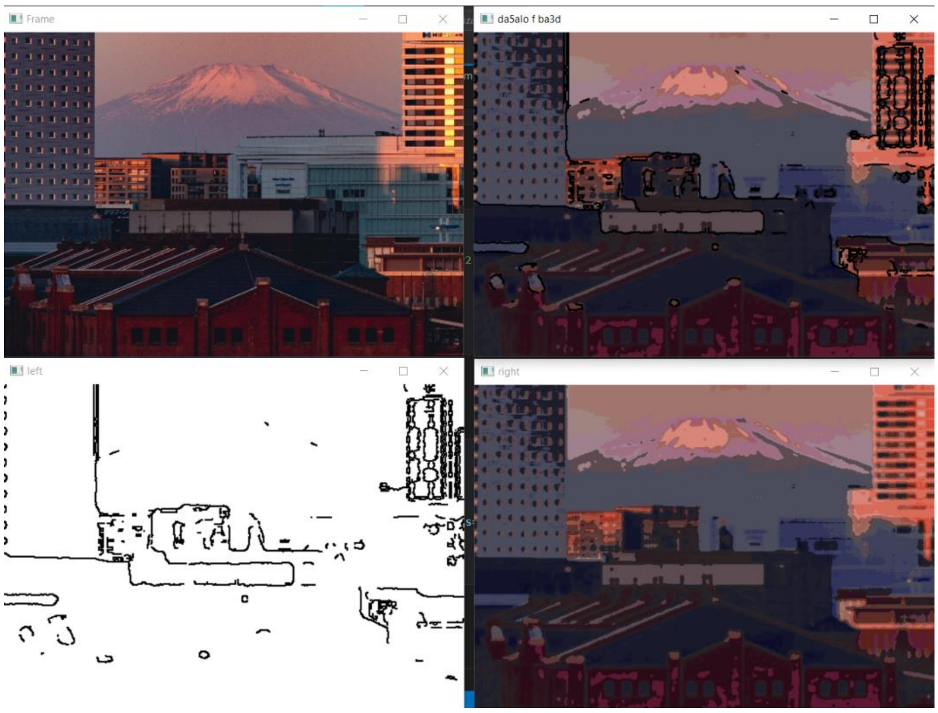
Results were rather underwhelming, but this could be contributed to the low number of classifiers (200) which we could not train more than that. We trained our model on an Azure machine, 128GB ram, 16 core. Yet we could only reach 200 classifier after 410 minutes (6:50 hours) of training. However, it was in fact faster than Haarcascade method.

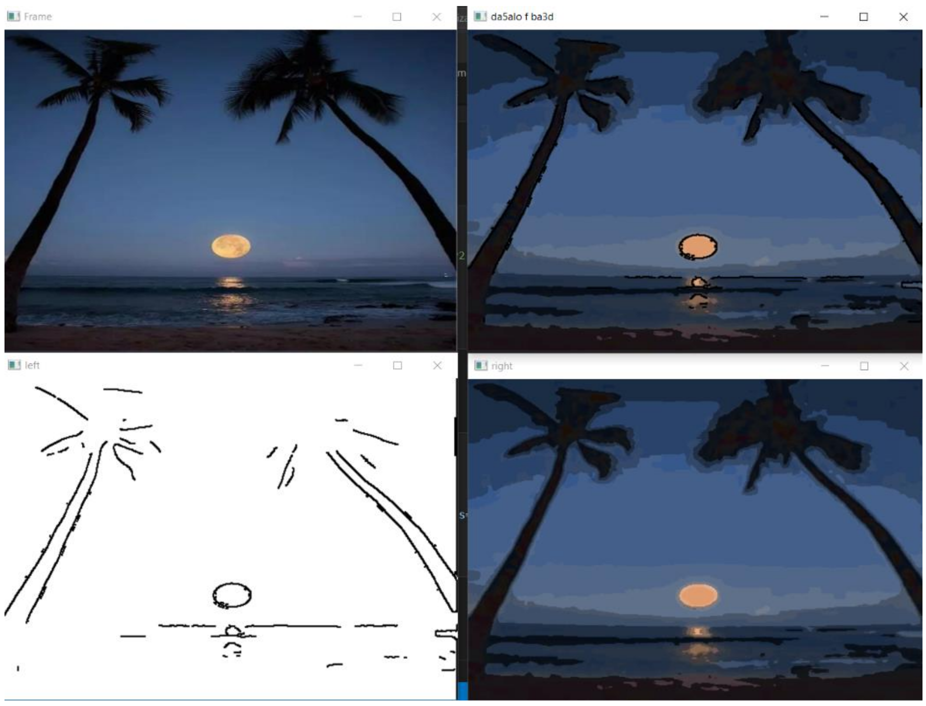
Quantization:

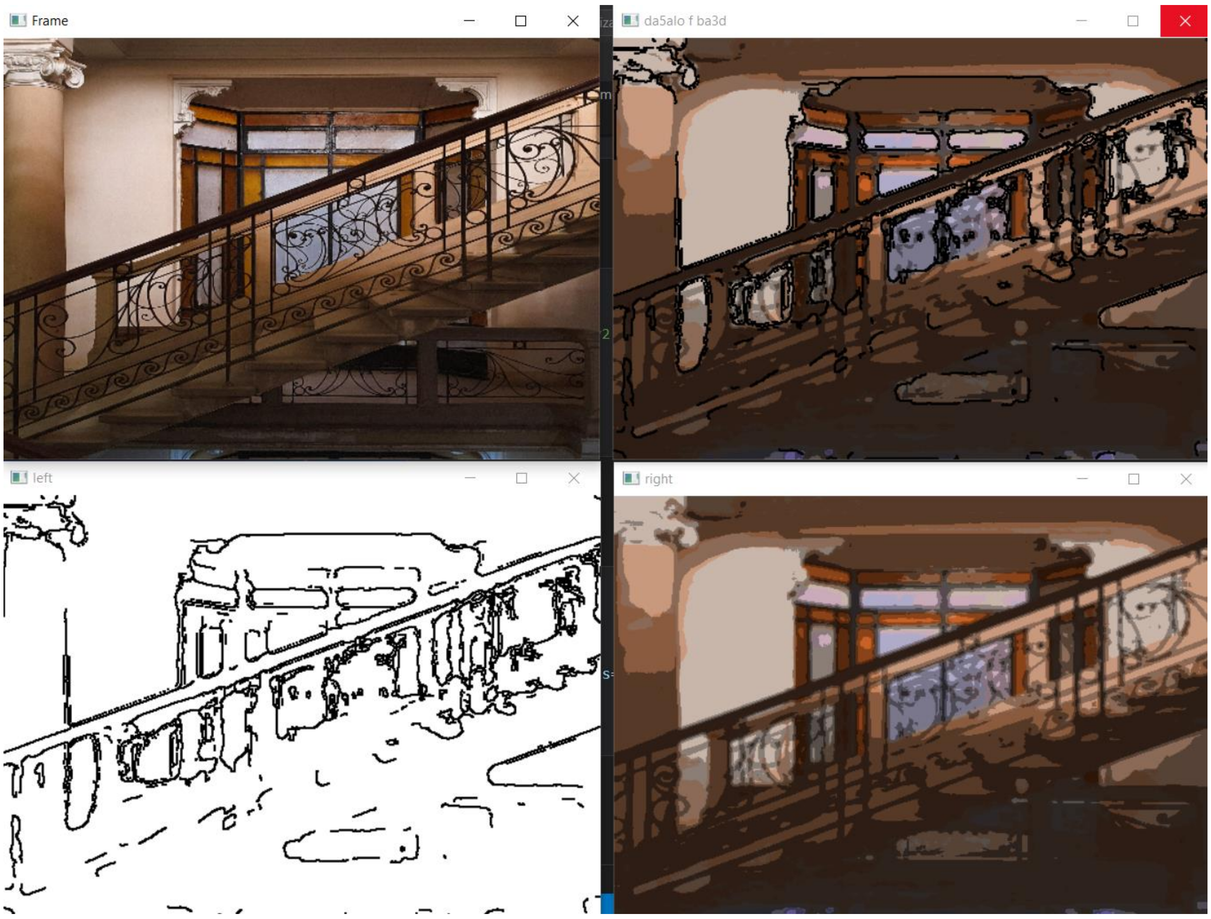
The three approaches each had their own pros and cons. The equation provided in the paper was rather simple and quick. However, it would often flicker in its colors due to a sudden change in the video that would push a dominant colour from one bin to another, causing massive changes in the displayed picture for a few frames before reverting back.

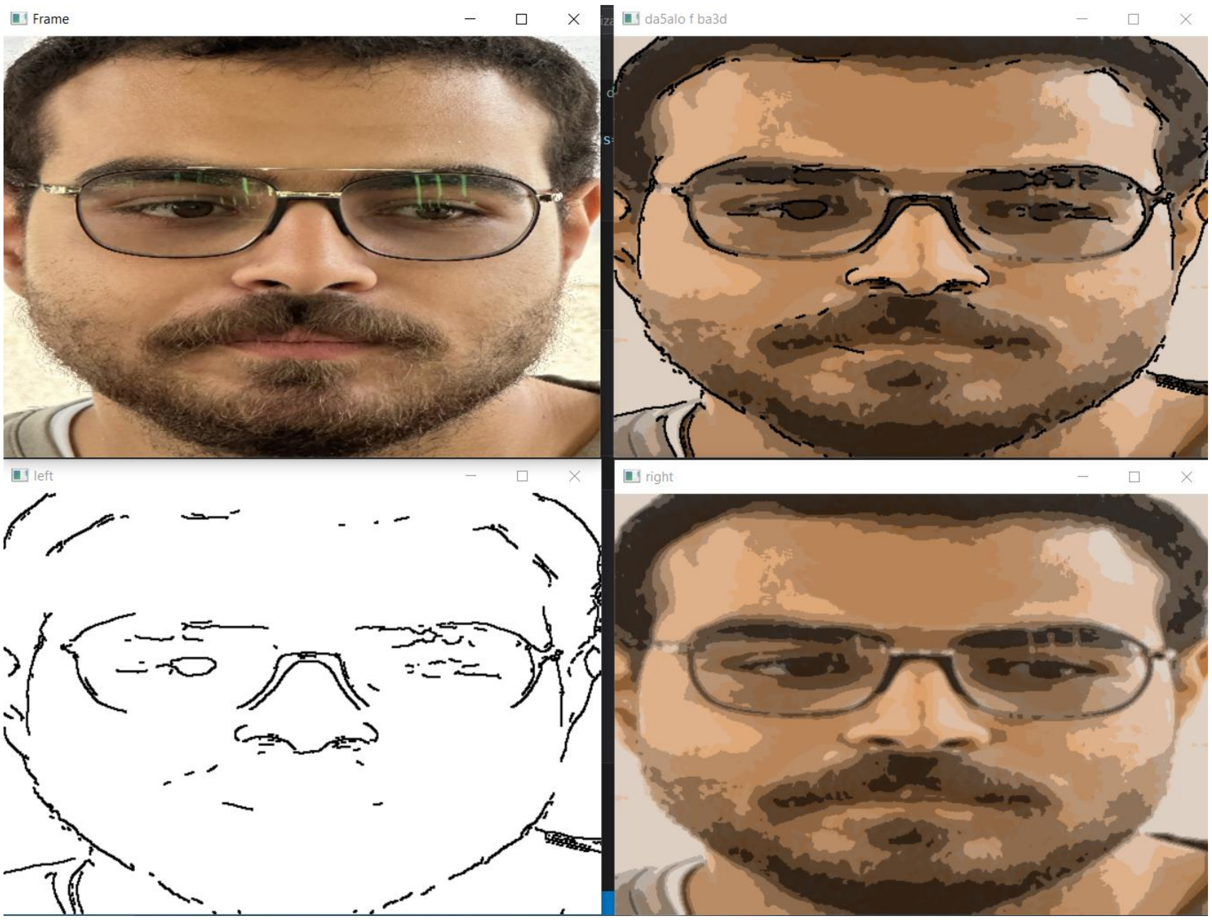
The Kmeans approach proved quite good, showing good quantization of colours that did not depend on pre-defined sets, unlike the previous equation. However, it would take more time to compute, that's why we opted for using it only with face detection, as it would work faster on a small Region of interest (ROI).

We then tried another Kmeans colour quantization approach, that calculated its own number of centroids dynamically. Through starting with one centroid and splitting it if it was deemed necessary (through a null hypothesis verification method). This showed the best results on big regions of pictures, but it was much slower than the rest. So, we decided to use it for pictures only, when cartoonizing the whole picture.

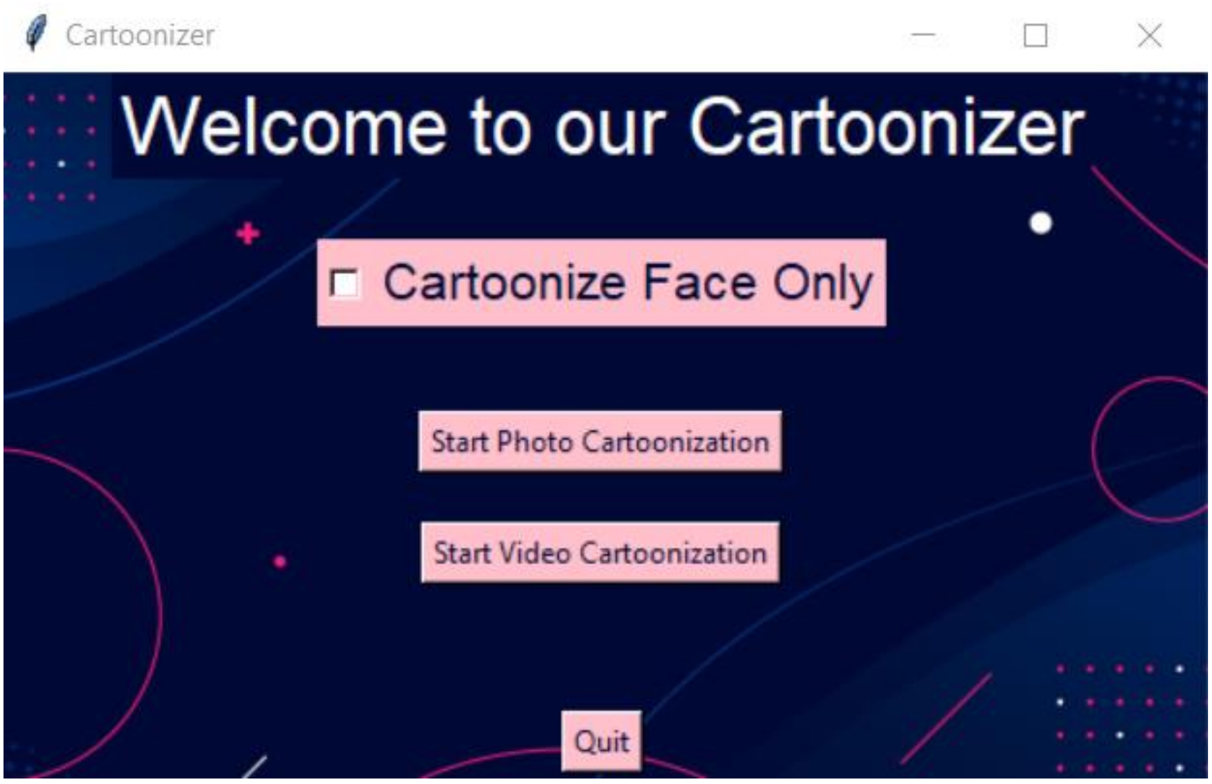





Frame
da5alo f ba3d
left
right

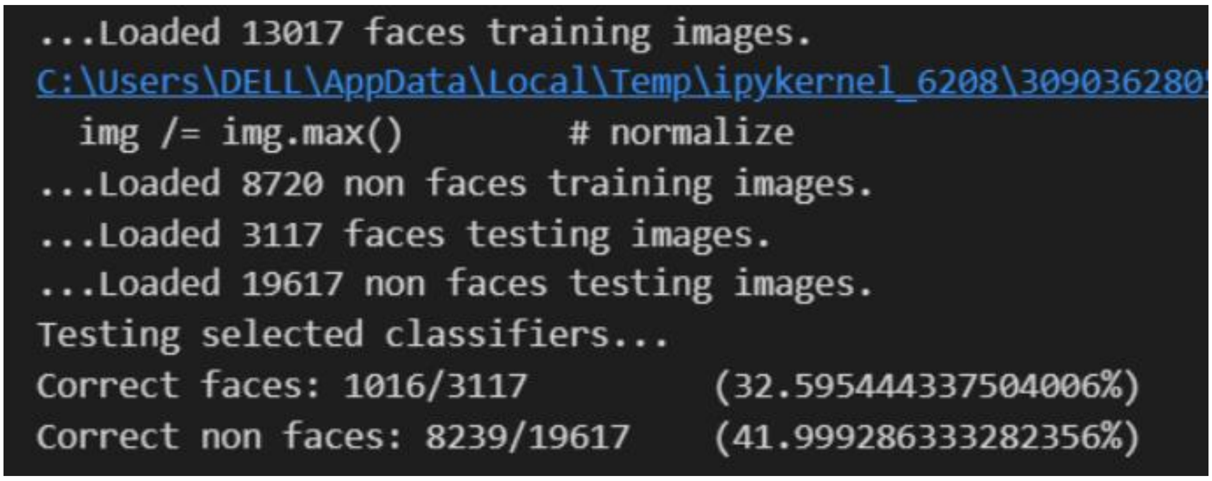

Frame
da5alo f ba3d
left
right

Fancy Responsive GUI


Cartoonizer
Welcome to our Cartoonizer
Cartoonize Face Only
Start Photo Cartoonization
Start Video Cartoonization
Quit

Performance & accuracy:

When training Viola Jones classifiers, we reached accuracy of 32% for faces, and 42% for nonfaces.



```
...Loaded 13017 faces training images.
C:\Users\DELL\AppData\Local\Temp\ipykernel_6208\30903628
  img /= img.max()        # normalize
...Loaded 8720 non faces training images.
...Loaded 3117 faces testing images.
...Loaded 19617 non faces testing images.
Testing selected classifiers...
Correct faces: 1016/3117         (32.595444337504006%)
Correct non faces: 8239/19617    (41.999286333282356%)
```

We used the accuracy metric, to show how many predictions were correct from all the predictions.

Members	Abdelrahman	Ali	Farah	Ziad
Tasks	Cartoonization Pipeline, Research	Viola Jones (60%), Research	Cartoonization Pipeline, Research	Viola Jones (40%), GUI Research

Additional Comment – Phase 2:

We offer real-time live video. This was done through optimizing the pipeline in case the user decides to choose this option. In which case, we use a quick quantizer, and also try to reduce the usage of complex computations.

Viola Jones accuracy can be greatly increased if more classifiers were chosen instead of 200. This was evident as the accuracy increased from 5% using 20 classifiers, to 22% using 100, then 27% using 150 and finally 32.5% for 200. We actually have more than 29,000 available classifiers that could be trained. However, this would require much more time, and needs a much stronger machine than even the one we used on Azure.

Conclusion & References:

In conclusion. We implemented 2 different pipelines. One using our own implementation of Viola Jones face detection algorithm, the other using built-in HaarCascade.

Viola Jones was faster, but Haar Cascade showed better results.

Furthermore, our project consists of two major pipelines. One which is responsible for edge detection, while the other for cartoonization. Lastly their outputs are added together to produce the final result. This is done using both approaches, and quantizing using 3 different approaches according to use case.

- S. K. Shrivastava and R. Gajjar, "Image Processing based Image to Cartoon Generation: Reducing complexity of large computation arising from Deep Learning," 2023 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES), Greater Noida, India, 2023, pp. 650-656, doi: 10.1109/CISES58720.2023.10183524.
- Viola, P. and Jones, M. (no date) 'Rapid object detection using a boosted cascade of Simple features', *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001* [Preprint]. doi:10.1109/cvpr.2001.990517.
- Dandan Che and Zhenjiang Miao, "Real-time cartoon style video generation," 2010 International Conference on Image Analysis and Signal Processing, Zhejiang, China, 2010, pp. 640-643, doi: 10.1109/IASP.2010.5476191.